LiveSync: Acoustic-based Live Video Synchronization

https://github.com/terry00123/LiveSync

Team 7

20160534 Taeckyung LEE
KAIST
terry00123@kaist.ac.kr

20170695 Junhyeok Choi
KAIST
junh1568@kaist.ac.kr

1 Introduction

With emerging interest on video-sharing platforms, live video streaming has been popular through platforms such as YouTube[1] or Twitch[2]. In the live video streaming context, multiple users can watch same video simultaneously with their own mobile devices.

However, due to the wireless network latency and diverse device characteristics, individuals' video timelines may slightly differ from each other to cause uncomfortable overlapping of sounds or video spoilers.

We present LiveSync, an automatic acoustic-based video synchronization tool that supports multiple users. LiveSync performs audio latency measurement *(Section 3.3)*, TDoA (Time Difference of Arrival) measurement *(Section 3.4)*, and propagation delay estimation *(Section 3.5)* to calculate manual delay to synchronize the audio output with existing sounds.

2 Related Work

2.1 Acoustic-based Video Synchronization

Audio content is the key feature that can possibly represent the time of the video. Previous researches [5, 7, 9, 10] used the raw audio content or calculated audio fingerprints to compare between multiple videos to synchronize them. However, all approaches use pre-recorded data, not real-time synchronization using both raw and recorded audio data.

2.2 Mobile Audio Synchronization

Mobile maestro [8] elaborates synchronization between multiple mobile devices to support MMA (Mobile Multispeaker Audio) applications. They implemented AMAC (Adaptive Mobile Audio Coordination) system to synchronize between multiple devices to produce rich and engaging surrounding sounds.

However, AMAC system is based on external communication scheme, that requires devices to be connected and communicate each other, to make inefficient to handle multiuser live video streaming. Also, the system requires the modification on the Android operating system to provide stable audio output API, thus has limited scalability.

[1]https://youtube.com
[2]https://twitch.tv

3 LiveSync

LiveSync is an automatic tool to synchronize live video streaming along multiple devices using acoustic-based approach. We explain main design goal and challenges, overview, and detailed technical components.

3.1 Design Goal & Challenges

To perform an acoustic-based live video synchronization, LiveSync assumes:

- All videos are playing in certain interval.
- Difference of distances does not exceed certain threshold as:

  $$\max_{j,k \in [1,N]-i \wedge j \neq k} |d_{ij} - d_{ik}| < 3.43, \forall i \in [1, N]$$

  to prevent time difference exceeding 10ms even after the synchronization[3].

By those assumptions, we set the main goal of LiveSync as:

- **Handy interaction**: Users should be easy to use functionalities without knowing detailed background or setting configurations.
- **Real-time**: Synchronization should be completed in reasonable time without interrupting live video watching experience.
- **Robustness**: External noises or diverse latency should not affect synchronization accuracy (time difference up to 10ms). Multiple devices should synchronize properly without significant error.

by referring the Knocker [6].

3.2 System Overview

LiveSync system is based on nature of mobile devices. As shown in Fig 1, to properly synchronize the audio content, we need to measure (i) Audio latency, (ii) TDoA (Time Difference of Arrival), (iii) Propagation delay, to calculate *Manual delay* to push/pull current device's live streaming.

After the synchronization, user can play the audio and broadcast itself to let other devices know itself has synchronized.

[3]The constant 3.43 is calculated by $343\text{m/s} \times 0.01\text{s}$. The equation means that none of the arrived sound pairs should make overlapping with the interval larger than 10ms.

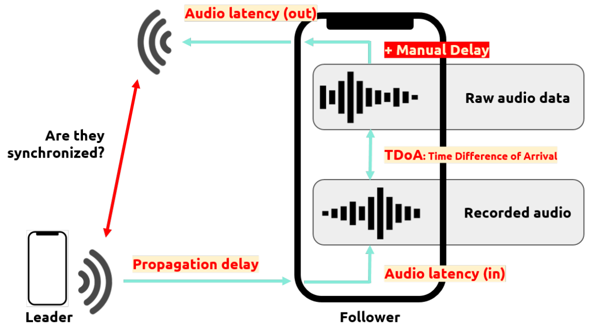

Figure 1. Overview of LiveSync system with two devices case.

3.3 Audio Latency

Since LiveSync handles audio input and output, audio latency problem should be considered properly [1]. All android devices have diverse audio latency per device, per execution. We define audio latency as round-trip latency which consists of input latency (for recorder), output latency (for speaker), and extra processing latency.

One tried to fix output latency [8] to generate stable audio output, however requires operating system level modification, so lack of scalability or deployability on existing devices.

Therefore, LiveSync repeatedly measures the audio latency [2] on the launch of the application. We first tried to use inaudible frequency to measure the latency, however our devices had failed to generate those frequency. Thus, we select frequencies in audible range. Final audio latency is the median of repeated measurements.

After measuring the audio latency, LiveSync continuously writes a zero-value buffer to the speaker to prevent needless sleep of the device driver [8].

3.4 Measuring TDoA

Prior researches [5, 7, 9, 10] utilized audio fingerprinting scheme to compare multiple audio data. Especially, off-the-shelf video editing program such as Adobe Premier utilizes audio fingerprinting with cross-correlation to synchronize multi-camera video data [5].

However, audio fingerprint requires additional computation to convert raw audio data into the fingerprint. Therefore, we adopt cross-correlation approach with raw audio data (one from recorder, the other from video resource). Finding TDoA by cross-correlation can be expressed as:

$$\tau_{\text{delay}} = \underset{t \in \mathbb{R}}{\arg\max}((f \star g)(t))$$

where $\star$ is a cross-correlation operation that can be speed up using FFT as:

$$f \star g = \mathcal{F}^{-1}\left\{\overline{\mathcal{F}\{f\}} \cdot \mathcal{F}\{g\}\right\}$$

where $\mathcal{F}$ denotes FFT, and over-line denotes the conjugation.

With 10 seconds of source audio and recorded audio, cross-correlation using FFT takes about 10 seconds to compute: which is same for recording time on the mobile device.

3.5 Estimating Propagation Delay

Speed of sound is a key factor of propagation delay. Due to the lack of temperature sensors, we assume room temperature (20 °C) for LiveSync operation. Speed of sound is about 343m/s in the room temperature.

To estimate the propagation delay, distance between two devices should be measured. Sound or BLE can be used, but we omit sound-based approach since it requires explicit communication and requires to know output audio latency, which cannot be measured or fixed by our approach. Therefore, we choose BLE to estimate the distance.

BLE-based distance estimation uses RSSI (Received Signal Strength Indicator) value. Then, the distance with RSSI can be described as follows [12]:

$$d = 10^{(A-R)/10N}$$

where d is the distance, R is the RSSI value, A is the RSSI value at 1 meter. N is the propagation constant that vary from 2 (free space model), to 4 (two-ray ground model)[4] [4, 11].

Also, to reduce the error of RSSI value, we applied Low-Pass Filter (LPF) as follows [12]:

$$R_n = \alpha R_{n-1} + (1-\alpha)T_n$$

where T_n is the measured RSSI value, R_n is the nth LPF-applied RSSI value, and α is the constant between 0 and 1. We use R_n as the RSSI value to estimate the distance and calculate propagation delay.

3.6 Putting All Together

With audio latency, TDoA, and propagation delay, we can now calculate the manual delay and add them into the sound output to synchronize with pre-existing sound.

As shown in Fig 2, our goal is to synchronize target device's audio output in same timeline with reference device's output. Equation (1) and (2) can be written as:

$$t + t_0 = t + t_1 + \Delta d - L_{out} \tag{1}$$

$$\text{TDoA} = (t + t_1) - (t + t_0 - \Delta t_{prop} - L_{in}) \tag{2}$$

Combining equation 1 and 2, we can calculate the manual delay Δd as:

$$\begin{aligned} \Delta d &= L_{out} - (t_1 - t_0) \\ &= L_{out} - (\text{TDoA} - \Delta t_{prop} - L_{in}) \\ &= L_{out} + L_{in} + \Delta t_{prop} - \text{TDoA} \\ &= \text{Audio\_Latency} + \text{Propagation\_delay} - \text{TDoA} \end{aligned} \tag{3}$$

[4]Free space model assumes the signal is propagated only in direct path; while two-ray model assumes the signal is propagated through one direct path and one reflected path.

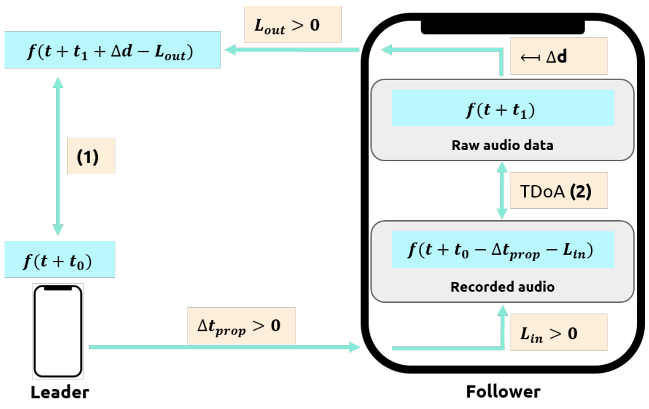

Figure 2. Putting all together. LiveSync calculates (1) and (2) to get manual delay Δd. Δt_{prop} is the propagation delay, and L is the audio latency.

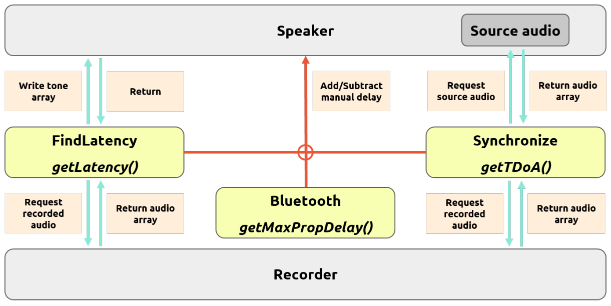

Figure 3. Overall LiveSync architecture. LiveSync consists of Recorder, Speaker, and Bluetooth module.

where $\Delta d > 0$ adds time to the current timeline and $\Delta d < 0$ subtracts time to the current timeline.

4 Implementation

Overall LiveSync architecture is shown in fig 3. Implementation details will be explained below.

4.1 Speaker & Recorder

Implemented in *Speaker.kt* for audio output, and *Recorder.kt* for audio input. Speaker (Recorder) is based on Android *AudioTrack* (*AudioRecord*) class with specialized thread to continuously write (read) audio data to prevent each module turning into the idle state.

4.2 Audio Latency

Implemented in *FindLatency.kt*. LiveSync generates the tone of randomly selected frequency in $f \in [9.5kHz, 10.5kHz]$, then measures the audio latency by calculating time difference between 1) writing tone to the Speaker, and 2) recognizing the frequency at the Recorder. LiveSync performs six repeated measurements to get median value for the final result.

Algorithm 1: Audio latency

```
freq = random(-95000, 105000);
S = {};
for $i$ = 1 to 6 do
    startTime = currentTimeInMillis();
    speaker.setSource(tone of freq);
    async
        speaker.play();
        array = recorder.recordWithTime();
    if freq in array then
        endTime = array.recordedTime;
        S.add(endTime- startTime);
return S.median();
```

4.3 Measuring TDoA

Implemented in *Synchronize.kt*. TDoA is measured by cross-correlation between source audio data and recorded audio data with duration of 10s for default. Cross-correlation is implemented in *CrossCorrelation.kt* with FFT.

Algorithm 2: Measuring TDoA

```
duration = 10000;
async
    contentArray = speaker.getSource(duration);
    recordedArray = recorder.record(duration);
return crossCorrelation(contentArray,
  recordedArray).argMax();
```

4.4 Estimating Propagation Delay

Implemented in *Bluetooth.kt*. When the device is synchronized, LiveSync marks device's Bluetooth name with 'SYNCHRONIZED' header[5]. Therefore, to estimate the propagation delay, LiveSync searches for marked devices nearby. If none exists, propagation delay is zero.

Algorithm 3: Estimating propagation delay

```
rssi = bluetooth.getMaxRSSI();
distance = 10.pow((A- rssi) / (10× N));
propDelay = distance × 1000 / 343;
return propDelay;
```

[5]Header can be extended to use the hash value of video metadata (ex. URL, title) to support synchronization among devices of different videos.

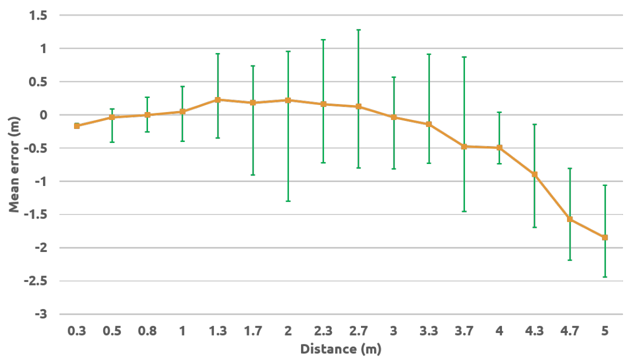

Figure 4. The figure shows the error of BLE-based distance estimation of 15 repeated evaluations. Error bars represent 10^{th} and 90^{th} percentiles.

Table 1. Constants for BLE-based distance estimation

Symbol	Value
A	-73.53
N	2.0
α	0.6

With the equation in Section 3.5, we conducted an experiment to check estimate accuracy and set each constant to use. Fig 4 represents BLE distance estimate error of 15 trials per distance. Since propagation delay of 2.5m is around 8ms, estimation errors are acceptable.

By the result of our experiment and previous research [12] with assuming free space propagation model [4, 11], we chose constants for the equation in Section 3.5 as Table 1.

5 Experiment

To evaluate the accuracy for further experiments, we used following criteria: "Is two sounds distinguishable in one-user's side?". Four Samsung Galaxy S7 of Android 8.0 were used for experiments. We also tried synchronization between Samsung Galaxy S7 Edge and Samsung Galaxy S8, and showed 95% accuracy. We believe LiveSync would operate among diverse device characteristics.

5.1 Impact of Distance

We evaluated the accuracy of LiveSync with the distance between two devices. Since exact coordination of video playing is impossible, we press one button then another; with range in 5s. We repeated evaluations 20 times per each distance.

As the result shown in Fig 5, LiveSync is robust to the distance up to 5m.

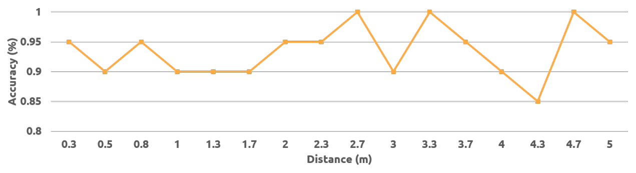

Figure 5. Accuracy of 20 repeated evaluations for each distance.

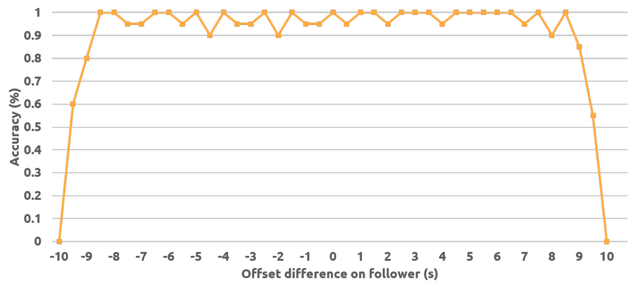

Figure 6. Accuracy of 20 repeated evaluation for each offset difference.

5.2 Impact of Offset Difference

Theoretically, LiveSync can recognize time offset difference $diff \in (-10s, 10s)$[6]. However, LiveSync may be inaccurate at the edge of range due to the short range of audio overlapping, which leads to the low accuracy of cross-correlation. We evaluated the impact of the *diff* for 20 times, with manually selecting video offset of t_0 at leader device and $t_0 + diff$ at follower device.

As the result shown in Fig 6, LiveSync is not accurate when $|diff| \geq 0.9$. Interestingly, only 1.5s overlapping segments over 10s interval was enough to detect the time difference.

5.3 Synchronizing with Multiple Devices

We measured the accuracy of synchronization with four devices: one leader and three followers. We conducted experiments in the dormitory room (Mir Hall, KAIST) with six different positions as Fig 7. As the result, we achieved 100% success rate of 20 experiments on each positions.

5.4 Impact of Type of the Audio

There are various types of the video as well as its audio content. Therefore, we measure the impact of audio type as in Fig 8.

As the result, audio type of music showed low accuracy, which indicates that music audio requires more accurate synchronization.

[6]Here, *diff* is different from TDoA since *diff* means the time difference on each device's timeline ($t_1 - t_0$) while TDoA contains propagation delay and input audio latency.

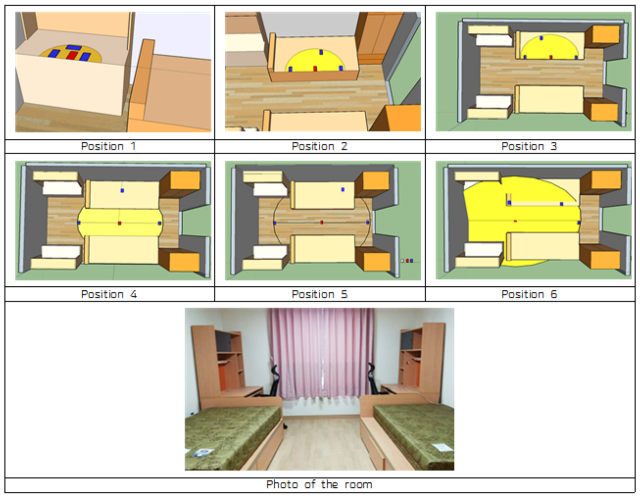

Figure 7. Six different positions of the multiple devices experiment. Blue square for follower devices and red square for a leader device.

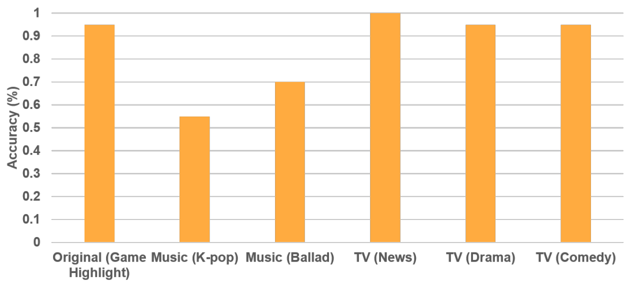

Figure 8. Accuracy of 20 repeated evaluations for each audio type.

5.5 Impact of Noise

Since users could use LiveSync in various situations (in quite place, with background music, with loud conversations), LiveSync should be robust to the noise. We conducted an experiment to play various noises as the background environment to operate LiveSync. The experiment is repeated 20 times per noise at 30cm away from the speaker with random time offset.

As the result shown in Fig 9, LiveSync showed over 90% accuracy for every type of the noise.

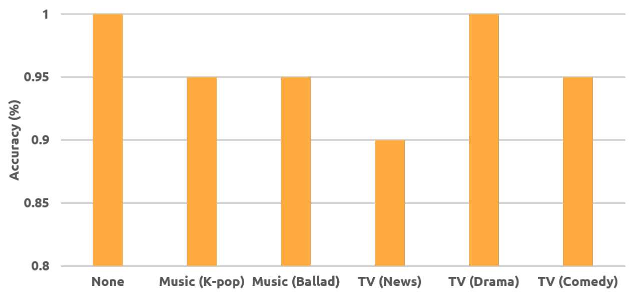

Figure 9. Accuracy of 20 repeated evaluations for each noise (None means no noise). Mean sound pressure level of each noise is measured as Table 2

6 Discussion

6.1 Applying on Existing Services

LiveSync shows sufficient accuracy and handy interactions with reasonable latency. However, applying LiveSync at off-the-shelf video streaming services is practically hard without knowing the source code of streaming service application. The main difficulty comes from gaining the source audio data.

Most of the video streaming services do not support audio extraction due to copyright issues. We manually extracted audio data from the raw video file and used as a separate input to solve the problem. Also, audio output of the device can only be captured using audio playback capture API [3] which is only available over Android 10.0 (Q) that most of the devices do not support. Therefore, devices under Android Q need to modify AOSP, which is not applicable to every existing devices.

6.2 Speedup

LiveSync requires 10s for audio recording, and another 10s for TDoA measurement. We believe that TDoA measurement with cross-correlation can boost up utilizing GPU resources in modern smartphones.

6.3 Utilizing Bluetooth on Android Devices

Currently, broadcasting Bluetooth on Android devices has limited broadcasting time up to 300 seconds. Also, broadcasting requires user popup for request. Therefore, to enable synchronized devices being searched, we need to repeatedly ask user for 300 seconds which may interrupt user's watching experience.

6.4 Energy Consumption & Privacy Concerns

For the fast response and stable audio latency, LiveSync requires always-on speaker and recorder. Also, LiveSync repeatedly checks nearby BLE devices using Bluetooth module. We measured battery usage of LiveSync and YouTube for 30 minutes, and both consumed 6% of the battery on our device. Considering LiveSync demo application does not include online connected live streaming (which YouTube has), LiveSync would consume at most twice energy than off-the-shelf live streaming applications.

Also, LiveSync requires always-on recorder so may cause privacy concerns. As showed in similar always-on recorder based research [6], users may accept using recorder if the application does not store or transfer the data. Since LiveSync

Table 2. Mean pressure level of the sound used in Fig 9. Measured at 15cm away from the speaker.

Type	None	Music (K-pop)	Music (Ballad)	TV (News)	TV (Drama)	TV (Comedy)
Mean sound pressure (dB)	31	59.3	54.1	43.5	42.1	44.5

only uses recorded data in cross-correlation, privacy can be protected.

7 Conclusion

LiveSync is an automatic acoustic-based live video synchronization tool. LiveSync operates accurately in 5m distance, with offset difference as 85% of synchronization duration, being robust to background noise. LiveSync showed relatively low accuracy for synchronizing music, which will be the future work.

Live streaming is an important technique that affects real user experience. We expect LiveSync would benefit real-life live video viewing experiences.

8 Contributions

- 20160534 Taeckyung LEE
  - Researched about BLE-based distance measurement.
  - Researched about TDoA measurement.
  - Implemented most of the Android application (Created *MainActivity.kt*, *Speaker.kt*, *Recorder.kt*, *Synchronize.kt*, *CrossCorrelation.kt*, and modified *FFT.kt* from the TA).
  - Planned experiments.

- 20170695 Junhyeok Choi
  - Researched about audio latency.
  - Implemented *FindLatency.kt* and *Bluetooth.kt*.
  - Planned & conducted experiments.
  - Created demo video.

9 What We Have Learned

Taeckyung: I really liked to work in our project for the semester. I think the hardest part was choosing the research topic. We did not have much experience of mobile computing; however I wanted to choose novel but real-life related topic. Thanks to our professor and TA for helping us to reach current topic.

Junhyeok: It was good to read and present the papers to get to know about the latest technologies in mobile computing. Actually, I was reluctant to read and present the papers in English. But through this experience, I grew up a lot and overcome my fear of speaking English. Moreover, as I worked on the project, I was able to figure out exactly what I have to study more to be a good programmer. Thanks to our professor and TA for helping our team, and my teammate Taekyung encouraged me.

Acknowledgments

Special thanks to professor Sukyoung Ryu at KAIST for providing mobile devices for our experiment, as well as professor Juhan Nam at KAIST for the advice on audio TDoA measurements. We would also like to thank our TA Jaemin Shin and professor Sung-Ju Lee for the full support.

References

[1] Android. 2019. Audio Latency. https://developer.android.com/ndk/guides/audio/audio-latency.html

[2] Android. 2019. Audio Latency Measurements. https://source.android.com/devices/audio/latency/measurements

[3] Android. 2019. Audio Playback Capture. https://developer.android.com/guide/topics/media/playback-capture

[4] Karl Benkic, Marko Malajner, P Planinsic, and Z Cucej. 2008. Using RSSI value for distance estimation in wireless sensor networks based on ZigBee. In *2008 15th International Conference on Systems, Signals and Image Processing*. IEEE, 303–306.

[5] Nicholas J Bryan, Paris Smaragdis, and Gautham J Mysore. 2012. Clustering and synchronizing multi-camera video via landmark cross-correlation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2389–2392.

[6] Taesik Gong, Hyunsung Cho, Bowon Lee, and Sung-Ju Lee. 2019. Knocker: Vibroacoustic-based Object Recognition with Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 3 (2019), 82.

[7] Lyndon Kennedy and Mor Naaman. 2009. Less talk, more rock: automated organization of community-contributed collections of concert videos. In *Proceedings of the 18th international conference on World wide web*. ACM, 311–320.

[8] Hyosu Kim, SangJeong Lee, Jung-Woo Choi, Hwidong Bae, Jiyeon Lee, Junehwa Song, and Insik Shin. 2014. Mobile maestro: enabling immersive multi-speaker audio applications on commodity mobile devices. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 277–288.

[9] Prarthana Shrestha, Mauro Barbieri, Hans Weda, and Dragan Sekulovski. 2009. Synchronization of multiple camera videos using audio-visual features. *IEEE Transactions on Multimedia* 12, 1 (2009), 79–92.

[10] Prarthana Shrstha, Mauro Barbieri, and Hans Weda. 2007. Synchronization of multi-camera video recordings based on audio. In *Proceedings of the 15th ACM international conference on Multimedia*. ACM, 545–548.

[11] Jiuqiang Xu, Wei Liu, Fenggao Lang, Yuanyuan Zhang, and Chenglong Wang. 2010. Distance measurement model based on RSSI in WSN. *Wireless Sensor Network* 2, 08 (2010), 606.

[12] Suk-Un Yoon, Jongha Woo, JungYon Cho, and Cheul-Ree Rahm. 2018. An Efficient Distance Estimation Method Based on Bluetooth Low Energy and Ultrasound. In *2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 206–212.